EXCEPTION HANDLING

1. SAMPLE -PROBLEM STATEMENT

"SmartHealth, a healthcare management platform, provides an interface for patients to input their health information. This system ensures the accuracy and security of the data entered. Upon validation, the system calculates health insurance premiums based on the user inputs. Robust exception handling is implemented to manage any erroneous inputs effectively.

Consider the following scenario:
You receive input including the patient's name, age, gender, height, weight, blood pressure, and cholesterol levels in the UserInterface class.
Component Specification: **HealthDataProcessor** (Utility class)

Type (Class)	Method	Parameters	Responsibilities
HealthDataProcessor	validatePatientDetails	int age, String gender, double height, double weight, int bloodPressure, double cholesterol	This method validates the patient details according to specified rules. If all details are valid, it returns true. If any detail is invalid, it throws an InvalidHealthDataException with the appropriate error message.

Validation Rules:

- **Age**: Must be between 18 and 100 inclusive. If not, throw an InvalidHealthDataException with the message "Invalid age".
- **Gender**: Must be either 'Male' or 'Female' (case sensitive). If not, throw an InvalidHealthDataException with the message "Invalid gender".
- **Height**: Must be greater than 0. If not, throw an InvalidHealthDataException with the message "Invalid height".
- **Weight**: Must be greater than 0. If not, throw an InvalidHealthDataException with the message "Invalid weight".
- **Blood Pressure:** Must be within the range of 60 to 220 mmHg. If not, throw an InvalidHealthDataException with the message "Invalid blood pressure".
- **Cholesterol:** Must be greater than or equal to 0. If not, throw an InvalidHealthDataException with the message "Invalid cholesterol".

HealthDataProcessor calculateInsurancePremium double height, double weight, int age, int bloodPressure, double cholesterol This method calculates and returns the health insurance premium based on the patient's health metrics. Guidelines for premium calculation are based on BMI, blood pressure, and cholesterol levels, with specific rates for different age groups and genders.

Component Specification: **InvalidHealthDataException** (This class inherits the Exception Class)

Type (Class)	Responsibilities
InvalidHealthDataException	Provided with a single-argument constructor: public InvalidHealthDataException(String message). It is thrown when age, gender, height, weight, blood pressure, or cholesterol levels do not follow the validation rules.

The main method in the UserInterface class starts by getting the patient's name, age, gender, height, weight, blood pressure, and cholesterol levels, and then calls the validatePatientDetails() method to validate these details. If the patient details are valid, the method calculates the insurance premium using the calculateInsurancePremium() method and prints the result. If any validation fails, an InvalidHealthDataException is caught, and the exception message is displayed.

Note:

- Propagate the exceptions that occur in the HealthDataProcessor class and handle them in the main method.
- In the sample input and output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.

calculateInsurancePremium() method (take some sample value and calculate the premium)

Example

if height,weight,bloodpressue and cholesterol very high , premium 1500
if height,weight,bloodpressue and cholesterol high , premium 1000
if height,weight,bloodpressue and cholesterol normal , premium 750

Sample Input/output 1:

Enter Name
Emma
Enter Age
25
Enter Gender
Female
Enter Height (in meters)
1.65
Enter Weight (in kg)
60
Enter Blood Pressure (mmHg)
120
Enter Cholesterol Level
5.2
Total Insurance Premium: $750.0

Sample Input/output 2:

Enter Name

John
Enter Age
16
Invalid age

==========

2. **SAMPLE -PROBLEM STATEMENT**

Banking Application

The banking application allows customers to manage their accounts online. Customers can log in using their credentials, perform transactions, and check their account balance. The system validates inputs to ensure security and correctness. Exception handling is used to manage errors effectively.

Component Specifications: AccountManager (Utility class)

Type (Class)	Method	Parameters	Responsibilities
AccountManager	validateCredentials	String username, String password	Validates the customer's username and password. Throws InvalidCredentialsException if invalid.
AccountManager	deposit	String username, double amount	Deposits the specified amount into the customer's account.
AccountManager	withdraw	String username, double amount	Withdraws the specified amount from the customer's account. Throws InsufficientFundsException if balance is insufficient.
AccountManager	checkBalance	String username	Returns the current balance of the customer's account.

UserInterface Class (Main method)

The main method in the `UserInterface` class prompts the user to enter their username and password to log in. If the credentials are valid, it allows the user to choose from options to deposit, withdraw, or check balance. Each action calls the appropriate method from `AccountManager` to perform operations on the account. Exceptions are handled to provide meaningful error messages to the user.

Additional Validation Conditions

1. Username and Password Validation:

- Username must be at least 6 characters long and can only contain alphanumeric characters and underscores.
- Password must be at least 8 characters long and should include at least one uppercase letter, one lowercase letter, one digit, and one special character.

2. Transaction Amount Validation:
- Deposit amount should be greater than zero and less than or equal to a predefined maximum deposit limit (e.g., $10,000).
- Withdrawal amount should be greater than zero and less than or equal to the current balance.

Explanation:

In this scenario, the `AccountManager` class handles operations related to account management such as validating credentials, depositing money, withdrawing money, and checking balances. Exceptions (`InvalidCredentialsException` and `InsufficientFundsException`) are used to handle errors gracefully and provide informative messages to the user via the `UserInterface` class.
Here's an expanded version with additional validation conditions and a more detailed set of sample inputs and outputs for the banking application scenario:

Additional Validation Conditions

1. Username and Password Validation:
- Username must be at least 6 characters long and can only contain alphanumeric characters and underscores.
- Password must be at least 8 characters long and should include at least one uppercase letter, one lowercase letter, one digit, and one special character.

2. Transaction Amount Validation:
- Deposit amount should be greater than zero and less than or equal to a predefined maximum deposit limit (e.g., $10,000).
- Withdrawal amount should be greater than zero and less than or equal to the current balance.

In the provided samples:

- **Sample 1:** demonstrates typical usage where deposits, withdrawals, and balance checks are performed within valid limits.
- **Sample 2:** includes additional validations:
  - It handles negative deposit attempts.
  - It prevents deposits exceeding a predefined maximum.
  - It handles withdrawal attempts that exceed the current balance.

These samples handling of user inputs and error conditions through the `AccountManager` and `UserInterface` classes. Each operation checks for validity and ensures that the user receives meaningful feedback through exceptions and informative messages.

By implementing these additional validation checks and error scenarios, the banking application becomes more secure and user-friendly, ensuring that transactions are processed safely and accurately.

Sample Input/Output

Sample Input 1:

```
Enter Username:
john_doe
Enter Password:
Password123!

Choose an Option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit

Enter Option:
1
Enter Amount to Deposit:
500
Deposit Successful. New Balance: $1500.00

Enter Option:
2
Enter Amount to Withdraw:
2000
Insufficient Funds. Current Balance: $1500.00

Enter Option:
3
Current Balance: $1500.00

Enter Option:
4
Logout Successful.
```

Sample Input 2:

```
Enter Username:
user123
Enter Password:
pass456!

Choose an Option:
```

1. Deposit
2. Withdraw
3. Check Balance
4. Exit

Enter Option:
1

Enter Amount to Deposit:
-100

Invalid amount. Please enter a positive number.

Enter Amount to Deposit:
15000

Deposit amount exceeds maximum limit of $10,000. Please enter a smaller amount.

Enter Amount to Deposit:
2000

Deposit Successful. New Balance: $2000.00

Enter Option:
3

Current Balance: $2000.00

Enter Option:
2

Enter Amount to Withdraw:
3000

Insufficient Funds. Current Balance: $2000.00

Enter Option:
4
Logout Successful.
```
==========

**3. SAMPLE -PROBLEM STATEMENT**

BookMyHotel is an online platform where users can book hotel rooms based on their preferences and dates of stay. The system requires users to log in using their credentials and validates their details before proceeding with the booking. Upon successful login and validation, the system calculates the total booking cost based on the selected hotel, room
```

type, duration of stay, and any additional preferences. Exception handling is implemented to manage any invalid inputs effectively.

Consider the following scenario:

You receive input that includes the user's name, age, email, and credit card details in the `UserInterface` class.

Component Specification: **HotelBooking** (Utility class)

Method: `validateUserDetails`
Parameters: `int age, String email, String creditCard`
Responsibilities: Validates user details according to predefined rules. Throws `InvalidBookingDetailsException` with appropriate messages if any detail is invalid.

Validation Rules:
- Age: Must be between 18 and 100 (inclusive). If not, throw an `InvalidBookingDetailsException` with the message "Invalid age".
- Email: Must be a valid email format (e.g., name@example.com). If not, throw an `InvalidBookingDetailsException` with the message "Invalid email".
- Credit Card: Must be exactly 16 digits. If not, throw an `InvalidBookingDetailsException` with the message "Invalid credit card number".

Method:`validateBookingDetails`
Parameters:`String hotelName, String roomType, LocalDate checkInDate, LocalDate checkOutDate`
Responsibilities: Validates booking details. Throws `InvalidBookingDetailsException` with appropriate messages if any detail is invalid.
Validation Rules:
- Hotel Name: Must be a recognized hotel name in the system. If not, throw an `InvalidBookingDetailsException` with the message "Invalid hotel name".
- Room Type: Must be either "Standard", "Deluxe", or "Suite" (case sensitive). If not, throw an `InvalidBookingDetailsException` with the message "Invalid room type".
- Check-in Date and Check-out-Date not in yyyy-MM-dd format throw an `InvalidBookingDetailsException` with the message "Invalid date format". (practice with DateTimerFormater(java8) and simpledateformate class)
- Check-in Date: Must be in the future. If not, throw an `InvalidBookingDetailsException` with the message "Invalid check-in date".
- Check-out Date: Must be after the check-in date. If not, throw an `InvalidBookingDetailsException` with the message "Invalid check-out date".

Method: `calculateBookingCost`
Parameters:`String roomType, int numNights, boolean breakfastIncluded`
Responsibilities: Calculates and returns the total booking cost based on the room type, duration of stay, and whether breakfast is included. Guidelines for cost calculation are based on provided rates for different room types and optional services.
Example (you can modify the below values according to the scenario)
Standard => 1000 per day

Deluxe=>2000 Per day
Suite =>3000 Per day
Breakfast =>100

Component Specification: **InvalidBookingDetailsException**

- **Responsibilities**: Inherits from `Exception`. It has a single-argument constructor: `public InvalidBookingDetailsException(String message)`. Thrown when any user or booking detail does not follow the validation rules.

The main method in the `UserInterface` class starts by getting the user's name, age, email, and credit card details, then calls `validateUserDetails()` to validate these details. If user details are valid, it proceeds to get the hotel name, room type, check-in, and check-out dates, validating these inputs using `validateBookingDetails()`. If all validations pass, the method calculates the total booking cost using `calculateBookingCost()` and prints the result. If any validation fails, an `InvalidBookingDetailsException` is caught, and the exception message is displayed.

Sample Input/Output:

```
Enter Name
John Doe
Enter Age
25
Enter Email
johndoe@example.com
Enter Credit Card Number
1234567890123456
Enter Hotel Name
Hilton
Select Room Type
Suite
Enter Check-in Date (yyyy-MM-dd)
2024-07-01
Enter Check-out Date (yyyy-MM-dd)
2024-07-05
Total Booking Cost: $3000.00
```
```
Enter Name
Jane Smith
Enter Age
17
Invalid age
```
```
Enter Name
```

Mike Johnson
Enter Age
30
Enter Email
mikejohnson@example
Invalid email
```
```
Enter Name
Sarah Lee
Enter Age
28
Enter Email
sarahlee@example.com
Enter Credit Card Number
12345
Invalid credit card number
```
```
Enter Name
Chris Brown
Enter Age
35
Enter Email
chrisbrown@example.com
Enter Credit Card Number
9876543210123456
Enter Hotel Name
Hyatt
Select Room Type
Penthouse
Invalid room type
```
```
Enter Name
Ella Davis
Enter Age
40
Enter Email
elladavis@example.com
Enter Credit Card Number
1111222233334444
Enter Hotel Name
Marriott
Select Room Type
Standard
Enter Check-in Date (yyyy-MM-dd)
2024-06-27

Enter Check-out Date (yyyy-MM-dd)
2024-06-25
Invalid check-out date
```
==========

4. **SAMPLE -PROBLEM STATEMENT**

**Healthcare Appointment Booking System**

Healthcare Hub, a leading healthcare provider, offers an online portal where patients can schedule appointments by providing their details. The system validates these details according to predefined rules to ensure accurate booking and patient information security. If the booking details are valid, the system schedules the appointment and confirms it. Exception handling is incorporated to manage invalid inputs effectively.

Scenario Description:

You receive input that includes the patient's name, age, phone number, health insurance ID, doctor's specialization, and appointment date/time in the `UserInterface` class.

Component Specifications: AppointmentBookingSystem (Utility class)

| Type (Class) | Method | Parameters | Responsibilities |
|---|---|---|---|
| AppointmentBookingSystem | alidatePatientDetails | int age, String phoneNumber, String insuranceId | Validates patient details (age, phone number, insurance ID) according to specified rules. Throws `InvalidAppointmentException` if any detail is invalid. |
| AppointmentBookingSystem | validateDoctorAvailability | String doctorSpecialization, LocalDateTime appointmentDateTime | Checks if the doctor specializing in the given field is available at the specified date and time. Throws `UnavailableDoctorException` if the doctor is not available. |
| AppointmentBookingSystem | scheduleAppointment | String patientName, String doctorSpecialization, LocalDateTime appointmentDateTime | Books the appointment for the patient with the specified doctor at the given date and time. |

***Validation Rules:***

**1. Patient Details:**
```

- **Age**: Must be between 1 and 120. If not, throw `InvalidAppointmentException` with the message "Invalid age".
- **Phone Number**:Must be exactly 10 digits. If not, throw `InvalidAppointmentException` with the message "Invalid phone number".
- **Insurance ID:** Must be alphanumeric. If not, throw `InvalidAppointmentException` with the message "Invalid insurance ID".

2. Doctor Availability:

- **Doctor Specialization**: Must match a predefined list of specialties (e.g., "Cardiology", "Dermatology", "Pediatrics"). If not, throw `UnavailableDoctorException` with the message "Doctor specialization not available".
- **Appointment Date/Time**: Must be in the future and within working hours. If not available, throw `UnavailableDoctorException` with the message "Doctor not available at specified time".

Sample Input/Output:

Sample Input/Output 1:

```
Enter patient name:
John Doe
Enter age:
45
Enter phone number:
9876543210
Enter insurance ID:
ABCD123456
Enter doctor specialization:
Cardiology
Enter appointment date/time (yyyy-MM-dd HH:mm):
2024-07-10 14:30

Appointment scheduled successfully with Dr. Smith (Cardiology) on 10th July 2024 at 2:30 PM.
```

Sample Input/Output 2:

```
Enter patient name:
Alice Smith
Enter age:
-5
Invalid age
```

Sample Input/Output 3:

```
Enter patient name:
Bob Johnson
Enter age:
35
Enter phone number:
12345
Invalid phone number
```

Sample Input/Output 4:

```
Enter patient name:
Eve Brown
Enter age:
28
Enter phone number:
8765432109
Enter insurance ID:
@#$%7890
Invalid insurance ID
```

Sample Input/Output 5:

```
Enter patient name:
Michael Lee
Enter age:
50
Enter phone number:
9876543210
Enter insurance ID:
XYZ4567890
Enter doctor specialization:
Neurology
Doctor specialization not available
```

Sample Input/Output 6:

```
Enter patient name:
Sophia Garcia
Enter age:
40
Enter phone number:
```

8765432109
Enter insurance ID:
PQR1234567
Enter doctor specialization:
Dentistry
Doctor not available at specified time
```

==========
```